09

# Para obtener más información: reutilizar entre constructores

En este capítulo, nuestro aprendizaje se centró en los constructores. Están diseñados para que los objetos tengan sus atributos inicializados en la construcción misma. Esta estrategia evita estados inconsistentes en nuestro objeto. Vea esta clase:

```
public class Carro{
    private int ano;
    private String modelo;
    private double precio;

    //getters e setters omitidos

}
```

COPIA EL CÓDIGO

Como ya se sabe, cuando el constructor no se declara en la clase, se utiliza el valor predeterminado, que no recibe ningún parámetro. Por lo tanto, un uso de la clase podría ser el siguiente:

```
Carro carro = new Carro();
carro.setAno(2013);
carro.setPrecio(35000.0);
```

COPIA EL CÓDIGO

¡Falta información valiosa! ¿Cuál es su modelo? Para evitar este tipo de problema, debemos exigir los datos que tienen sentido cuando se crea el carro. Algo como:

```
public class Carro{
    private int ano;
    private String modelo;
    private double precio;

    public Carro(int ano, String modelo, double precio){
        this.ano = ano;
        this.modelo = modelo;
        this.preco = precio;
    }

    //getters y setters omitidos

}
```

COPIA EL CÓDIGO

Ahora el uso requiere la presencia de los 3 parámetros definidos.

```
Carro carro = new Carro(2013, "Gol", 35000.0);
```

COPIA EL CÓDIGO

¡Todo funciona bien! Hasta que un día se le pide a nuestro sistema que acepte la creación con el paso de solo el modelo y el valor. En esta situación, el año debería verse como 2017. Una solución sería:

```
public class Carro{
    private int ano;
    private String modelo;
    private double precio;
```

```
    public Carro(int ano, String modelo, double precio){
        this.ano = ano;
        this.modelo = modelo;
        this.precio = precio;
    }

    //Nuevo constructor AQUI!
    public Carro(String modelo, double precio){
        this.ano = 2017;
        this.modelo = modelo;
        this.precio = precio;
    }

    //getters e setters omitidos

}
```

COPIA EL CÓDIGO

Y de esa manera puedes construir autos con cualquiera de los dos constructores:

```
Carro carro = new Carro(2013, "Gol", 35000.0);
Carro otroCarro = new Carro("Civic", 95000.0);
```

COPIA EL CÓDIGO

Sin embargo, en la compañía donde se codifica este sistema, hay un equipo de prueba que verificó que nuestro sistema permite la creación de un automóvil con fechas anteriores al primer automóvil que llegó a Brasil, un Peugeot traído por Santos Dumont en 1891. (Alura también ¡es historia!) Además de permitir que el modelo no se pase (null) y el precio inválido.

El desarrollador pronto intentó implementar esta regla en uno de los constructores:

```
public class Carro{
    private int ano;
    private String modelo;
    private double precio;

    public Carro(int ano, String modelo, double precio){
        if(ano >= 1891){
            this.ano = ano;
        }else{
            System.out.println("El año informado no es valido.
            this.ano = 2017;
        }

        if( modelo != null){
            this.modelo = modelo;
        }else{
            System.out.println("El modelo no fue informado. Por
            this.modelo = "Gol";
        }

        if(precio > 0){
            this.precio = precio;
        }else{
            System.out.println("El precio no es valido. Por es
            this.precio = 40000.0;
        }
    }
    //....
```

```
}
```

COPIA EL CÓDIGO

Tenga en cuenta que, como tenemos dos constructores, la regla también debería aplicarse al otro:

```
public class Carro{
    private int ano;
    private String modelo;
    private double precio;

    public Carro(int ano, String modelo, double precio){
        if(ano >= 1891){
            this.ano = ano;
        }else{
            System.out.println("El año informado no es válido.
            this.ano = 2017;
        }

        if( modelo != null){
            this.modelo = modelo;
        }else{
            System.out.println("El modelo no fue informado. Por
            this.modelo = "Gol";
        }

        if(precio > 0){
            this.precio = precio;
        }else{
            System.out.println("El precio no es válido. Por es
            this.precio = 40000.0;
```

```
    }

}

//Nuevo construtor AQUI!
public Carro(String modelo, double precio){
    this.ano = 2017;
    if( modelo != null){
        this.modelo = modelo;
    }else{
        System.out.println("El modelo no fue informado. Por
        this.modelo = "Gol";
    }

    if(precio > 0){
        this.precio = precio;
    }else{
        System.out.println("El precio no es válido. Por es
        this.precio = 40000.0;

    }

    //getters e setters omitidos

}
```

COPIA EL CÓDIGO

¡Funcionó, pero el código está duplicado y nuestra clase comienza a no verse bien! Los códigos duplicados requieren un mantenimiento doble en el futuro y, en la mayoría de los casos, un futuro no muy lejano. Sería genial si fuera posible reutilizar la lógica de validación del primer constructor declarado, ¿no? Reutilizaríamos todo y cualquier cambio también tendría un impacto directo.

En Java podemos llamar a la implementación de un constructor a través de otro usando simplemente `this ()` con los parámetros requeridos por el constructor.

Observe cómo se vería el segundo constructor de nuestra clase:

```
public Carro(String modelo, double precio){
    //llamando al constructor que recibe int, String y double
        this(2017, modelo, preco);
    }
```

COPIA EL CÓDIGO

Mucho más simple de mantener, ¿no? Nuestra clase, Carro, se vería así:

```
public class Carro{
    private int ano;
    private String modelo;
    private double precio;

    public Carro(int ano, String modelo, double precio){
        if(ano >= 1891){
            this.ano = ano;
        }else{
            System.out.println("El año informado no es válido.
            this.ano = 2017;
        }

        if( modelo != null){
            this.modelo = modelo;
        }else{
            System.out.println("El modelo no fue informado. Po
```

```
            this.modelo = "Gol";
        }

        if(precio > 0){
            this.precio = precio;
        }else{
            System.out.println("EL precio no es válido. Por es
            this.precio = 40000.0;
        }
    }

    //Nuevo constructor AQUI!
    public Carro(String modelo, double precio){
        this(2017, modelo, precio);
    }

    //getters e setters omitidos

}
```

COPIA EL CÓDIGO

## Conclusión

En Java, es posible llamar a un constructor dentro de otro, y esto se hace para evitar la duplicación de códigos y reglas. Después de todo, una regla aplicada en un constructor normalmente será la misma para el otro caso. Para esto, se usa `this()`, pasando los parámetros correspondientes al constructor al que desea llamar.